JCSI 3 (2017) 54-58
Journal of Computer Sciences Institute

Wysłane: 2016-12-15
Przyjęte: 2016-12-16

Porównanie aplikacji mobilnej w językach Swift i Objective-C

Kacper Erwin Sienkiewicz*, Edyta Łukasik
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Tematyką artykułu jest porównanie metodyk wytwarzania aplikacji mobilnych w językach Swift i Objective-C. W ramach przeprowadzonej analizy zostaną wskazane podobieństwa i różnice w implementacji aplikacji na te dwa języki programowania. Zaprojektowana została i zaimplementowana aplikacja Magic Drawing Board wykorzystująca silnik QUARTZ 2D. Stworzono dwie identyczne funkcjonalnie aplikacje. Analiza porównawcza została przeprowadzona dopiero po dokładnym objaśnieniu implementowanych widoków.

Słowa kluczowe: metody wytwarzania aplikacji; analiza implementacji widoków; silnik QUARTZ 2D.

*Autor do korespondencji.

Adres e-mail: ksienkiewicz14@gmail.com

Comparison of mobile application using Swift and Objective-C

Kacper Erwin Sienkiewicz*, Edyta Łukasik
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The subject of the article is to compare the methodologies for the production of mobile applications in Swift and Objective-C languages. The similarities and differences of the implementation of applications for these two programming languages will be identified as a part of the analysis. An Magic Drawing Board application using Quartz 2D engine was designed and implemented. Two identical functional applications were created. The comparative analysis was carried out only after a thorough explanation of the implemented views.

Keywords: method of applications development; the analysis of the implementation of views; QUARTZ 2D engine.

*Corresponding author.

E-mail address: ksienkiewicz14@gmail.com

1. Wstęp

W przeciągu ostatnich dekad znacznie wzrosło znaczenie aplikacji mobilnych w technologiach informatycznych. Obecnie większość populacji w krajach dobrze rozwiniętych nie wyobraża sobie funkcjonowania w życiu codziennym bez smartfonów i zawartych w nim aplikacji. Głównie dotyczy to prostych rzeczy, takich jak: sprawdzenie rozkładu jazdy autobusu, sprawdzenia pogody, czy kontakt z innymi użytkownikami przez media społecznościowe. Za tymi postępami musi również nadążać technologia, dlatego w dziedzinie programistycznej również widać, jak z każdym rokiem pojawią się nowe języki programowanie lub modernizacje języków starszej daty.

Firma Apple, znana z renomy swoich produktów, nie mogła dłużej czekać na wypuszczenie nowego języka. Podczas konferencji organizowanej przez firmę Apple o nazwie *Worldwide Developers Conference* (WWDC), 2 czerwca 2014 został zaprezentowany język Swift. Aktualna wówczas wersja języka była to *Swift 1.0*. Ogólnym założeniem języka jest zastąpienie Objective-C. Ma on służyć do sprawniejszego niż dotąd tworzenia aplikacji pracujących pod kontrolą systemów *OS X* i *iOS*. Aktualnie Swift ciągle się rozwija i na każdej corocznej czerwcowej konferencji *WWDC* pojawia się inna wersja języka. W 2016 roku była to wersja *Swift 3.0*. Ostatnia wersja języka wprowadziła bardzo wiele zmian w składni, jeszcze bardziej podkreślając, jak jest to wysokopoziomowy język programowania [1].

Apple stworzyło Swift z myślą o obniżeniu bariery wejścia dla programistów chcących tworzyć oprogramowanie w ich technologii. Nowy język jest tym narzędziem, do którego przyzwyczajeni są młodzi programiści, z wysokopoziomowymi strukturami wbudowanymi w język i wieloma ustawieniami systemowymi.

Celem artykułu jest porównanie metodyk wytwarzania oprogramowania na stworzonej aplikacji Magic Drawing Board w językach Swift i Objective-C. Analizie porównawczej została poddana składania obu tych języków. Podstawowa funkcjonalność tej aplikacji to edytor do rysowania z możliwością zapisu obrazu w galerii zdjęć. Aplikacja została zaprojektowana na urządzenie iPad.

W celu utworzenia aplikacji mobilnej wykorzystano następujące narzędzia:

- środowisko programistyczne X-code [2];
- języki programowania Swift i Objective-C;
- silnik QUARTZ 2D;
- emulatory oraz urządzenie iPad Pro.
- Do porównania obu języków została przeprowadzona analiza teoretyczna związana z funkcjonowaniem metod implementujących widoki aplikacji. W jej wyniku wskazane zostały podobieństwa i różnice równocześnie mocne i słabe strony każdego z języków. Zwrócono głównie uwagę na najważniejsze aspekty związane z działaniem obu aplikacji, czyli:
- składania języków;

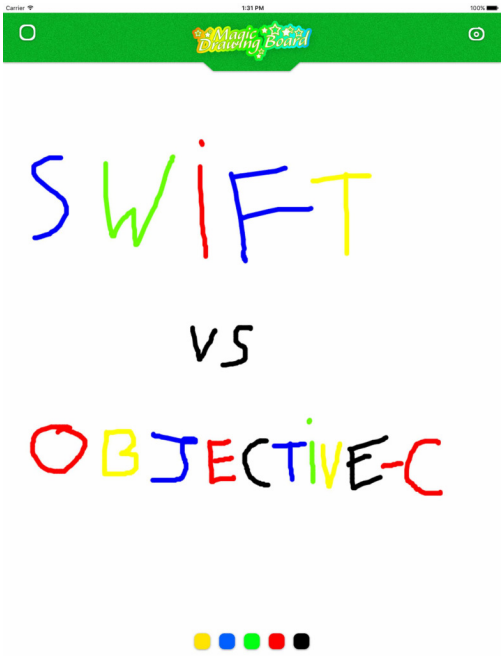
- wykorzystane metody;
- wykorzystanie silnika QUARTZ 2D.

2. Opis aplikacji

Utworzone zostały dwie identyczne aplikacje pod względem graficznym. Każda z nich posiada następujące funkcjonalności:

- w lewym górnym rogu znajduje się przycisk, który służy do wyczyszczenia wszystkiego, co zostało narysowane;
- przycisk po prawej stronie zapisuje narysowany obrazek i wysyła go do wbudowanej na urządzeniu aplikacji *zdjęcia*;
- pięć dostępnych przycisków reprezentujących kolory.

Wygląd przykładowych ekranów działających aplikacji w obu językach został pokazany na rysunkach 1 i 2. Po uruchomieniu aplikacji pojawi się główny widok aplikacji. Przykładowy efekt jej działania pokazano na rys. 1.



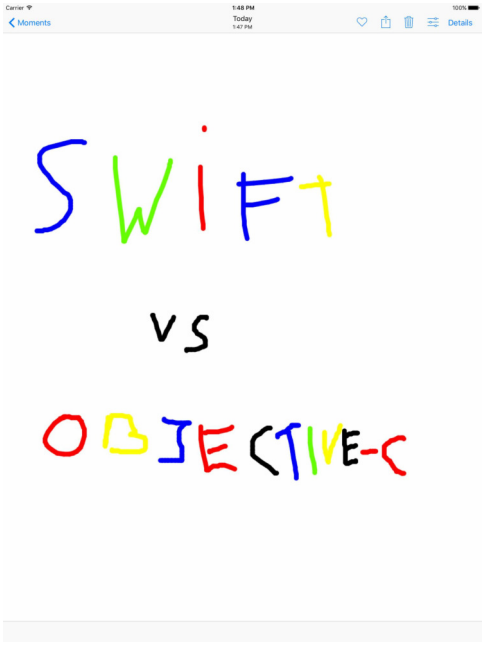
Rys. 1. Interfejs graficzny w aplikacji Magic Drawing Board dla języków Swift i Objective-C

Zapisany, po naciśnieciu przycisku umożliwiającego zapis aktualnego stanu ekranu, w galeri zdjęć narysowany obraz pokazano na rys. 2.

3. Opisanie składni języków Swift i Objetive-C na przykładzie listingów

3.1 Opis użytych funkcji w języku Swift

Funkcja *touchesBegan*, przedstawiona na przykładzie 1, odpowiedzialna jest za początek rysowania linii na ekranie [3]. Słowo kluczowe *override* umieszczone przez nazwą funkcji oznacza, że jest to metoda domyślna, która generuje automatycznie kod. W środku funkcji jest zainicjalizowanie zmiennej old, w której zawarte zostanie pierwsze dotknięcie oraz lokalizacja na widoku ekranu.



Rys. 2. Zapisany stan obrazka w galerii zdjęć

Przykład 1. Implementacja dotyku użytkownika na ekranie

```
var old:CGPoint?
override func touchesBegan(_ touches: Set<UITouch>, with
event: UIEvent?)
{
      old = touches.first?.location(in: self.view)
 }
```

Kolejna domyślna funkcja, *touchesMoved* określa, w którym kierunku podąża palec użytkownika. Wewnątrz metody zmienna *current*, również pełni tę samą rolę, co zmienna *old,* lecz jest wykorzystana w celu określenia, w którym kierunku zmierza rysowana linia. Następna zmienna o nazwie *ctt* reprezentuje obiekt *UIGraphics-GetCurrentContext*, który z założenia jest prostokątem. Następnie jest wpisana nazwa zmiennej *ctt* i odwołania po kropce do właściwości tej figury i podane są własne ustawienia. Funkcja ta przedstawiona została na przykładzie 2.

Przykład 2. Implementacja ruchu rysowanej linii

```
override func touchesMoved(_ touches: Set<UITouch>, with
event: UIEvent?) {
      let current = touches.first?.location(in: self.view)
      let ctt = UIGraphicsGetCurrentContext()
      ctt?.setStrokeColor(currentcolor.cgColor)
      ctt?.setLineWidth(10.0)

      ctt?.move(to: CGPoint(x: old!.x, y: old!.y))
      ctt?.addLine(to: CGPoint(x: current!.x, y: current!.y))
      ctt?.strokePath()

      let image =
UIGraphicsGetImageFromCurrentImageContext()
      (self.view as! UIImageView).image=image
}
```

W celu oprogramowania przycisku *zapisz w galerii,* wewnątrz funkcji została wykorzystana domyślna metoda *UIImageWriteToSavedPhotosAlbum.* Umożliwia ona w prosty sposób przekazanie do aplikacji *zdjęcia* tego, co

zostało narysowane na ekranie[4]. Jej zawartość przedstawiono na przykładzie 3.

Przykład 3. Implementacja przycisku zapisującego rysunek w galerii zdjęć

```
@IBAction func saveCamera(_ sender: AnyObject) {
UIImageWriteToSavedPhotosAlbum(UIGraphicsGetImageFro
mCurrentImag
eContext()!, nil,nil,nil)
}
```

W każdej funkcji jest odwołanie do zmiennej *currentcolor*, która przyjmuje wartość obiektu typu *UIColor*, Zapewnia ona dostęp do podstawowych kolorów, w tym przypadku jest to *red* (czerwony). Ustala ona jakiego koloru będzie rysowana pierwsza linia. Jej użycie pokazano na przykładzie 4.

Przykład 4. Oprogramowanie dwóch z pięciu kolorowych przycisków

```
var currentcolor=UIColor.red
 @IBAction func yellowClicked(_ sender: AnyObject) {
     currentcolor=UIColor.yellow
 }
   @IBAction func blueClicked(_ sender: AnyObject) {
      currentcolor=UIColor.blue
 }
```

Przycisk *clean* wywołuje funkcje *initContext,* której zawartość pokazano na przykładzie 5. Jest to metoda, która zawiera w sobie obiekt *UIGraphicBeginImageContext*, który sam w sobie zawiera funkcje umożliwiającą zresetowanie widoku *UIImageView*.

Przykład 5. Wywołanie funkcji initContext w implementacji przycisku cleanClicked

```
@IBAction func cleanClicked(_ sender: AnyObject) {
      self.initContext()
   }
func initContext() {
      UIGraphicsBeginImageContext(self.view.frame.size)
      let image =
UIGraphicsGetImageFromCurrentImageContext()
      (self.view as! UIImageView).image=image
   }
```

3.2 Opis użytych metod w języku Objective-C

W pliku ViewController.h, przedstawionym na przykładzie 6, dodane zostały instancje zmiennych:

- *lastPoint* – przechowuje ostatni namalowany punkt na obrazie;
- *red*, *green*, *blue* – zapisuje aktualną wartość RGB wybranego koloru;
- *brush*, *opacity* – przechowują szerokość pociągnięcia pędzla i jego krycie;
- *mouseSwiped* – identyfikuje, czy pociągnięcie pędzla *brush* będzie kontynuowane.

Przykład 6. Zawartość klasy ViewController.h

```
@interface ViewController : UIViewController {
   CGPoint lastPoint;
   CGFloat red;
   CGFloat green;
   CGFloat blue;
   CGFloat brush;
   CGFloat opacity;
}
@property (weak, nonatomic)
IBOutlet UIImageView *mainImage;
- (IBAction)pencilPressed:(id)sender;
- (IBAction)save:(id)sender;
- (IBAction)reset:(id)sender;
@end
```

Następnie zostały ustawione połączenia dla widoków utworzonych w *Interface Builder*. Obiekt typu *UIImageView* o nazwie *mainImage* to główny obraz, na nim odbywa się rysowanie.

Utworzone zostały połączenia dla pięciu kolorów, dla przycisków: *pencilPressed, save* i *reset*. Wartość początkowa koloru w notacji *RGB* jest ustawiona na czarno. Domyślne krycie jest ustawione na 1.0, a szerokość linii na 10.0. Na starcie aplikacji zostaną te ustawienia od razu zastosowane, gdyż znajdują się w metodzie *viewDidLoad*, pokazanej na przykładzie 7 [5].

Przykład 7. Metoda wywołana na starcie aplikacji

```
- (void)viewDidLoad {
   red = 0.0/255.0;
   green = 0.0/255.0;
   blue = 0.0/255.0;
   brush = 10.0;
   opacity = 1.0;
   [super viewDidLoad];
}
```

W metodzie *touchesBegan* na przykładzie 8, zmienna *lastPoint* jest inicjalizowana wartością bieżącego punktu dotyku. Metoda lokalizuje, w którym miejscu użytkownik dotknął ekranu.

Przykład 8. Implementacja metody touchesBegan

```
- (void)touchesBegan:(NSSet *)touches withEvent:
 (UIEvent *)event {
   UITouch *touch = [touches anyObject];
   lastPoint = [touch locationInView:self.view];
}
```

Metoda *touchesMoved* śledząca ruch dotyku pokazana została na przykładzie 9. Na początku metody zostanie zlokalizowana pozycja dotyku na ekranie. Pozwoli to następującym obiektom: *CGContextMoveToPoint*, *CGContextAddLineToPoint* i *CGContextSetLineCap* na narysowanie linii z ostatniego zlokalizowanego punktu dotyku do bieżącego [6].

Przykład 9. Implementacja metody touchesMoved

```
- (void)touchesMoved:(NSSet *)touches withEvent:
(UIEvent *)event {
   mouseSwiped = YES;
   UITouch *touch = [touches anyObject];
   CGPoint currentPoint = [touch locationInView:self.view];
   UIGraphicsBeginImageContext(self.view.frame.size);
   [self.mainImage.image drawInRect:CGRectMake(0, 0,
   self.view.frame.size.width, self.view.frame.size.height)];
   CGContextMoveToPoint(UIGraphicsGetCurrentContext(),
   lastPoint.x, lastPoint.y);
   GContextAddLineToPoint(UIGraphicsGetCurrentContext(),
   currentPoint.x, currentPoint.y);
   CGContextSetLineCap(UIGraphicsGetCurrentContext(),
   kCGLineCapRound);
```

Kolejne trzy obiekty: *CGContextSetLineWidth*, *CGContextSetRGBStrokeColor* i *CGContextSetBlendMode* ustawiają rozmiar pędzla, jego poziom krycia oraz kolor. Ich użycie pokazano na przykładzie 10.

Przykład 10. Inicjalizacja obiektów w metodzie touchesMoved

```
CGContextSetLineWidth(UIGraphicsGetCurrentContext(),
 brush );
CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext
(), red, green, blue, 1.0);
CGContextSetBlendMode(UIGraphicsGetCurrentContext(),
kCGBlendModeNormal);
```

Obiekt CGContextStrokePath, którego użycie pokazano na przykładzie 11, zakończy działanie metody poprzez narysowanie ścieżki na obrazie.

Przykład 11. Zakończenie metody touchesMoved

```
CGContextStrokePath(UIGraphicsGetCurrentContext());
   self.mainImage.image=
 UIGraphicsGetImageFromCurrentImageContext();
   [self.mainImage setAlpha:opacity];
   UIGraphicsEndImageContext();
}
```

Do każdego przycisku został przypisany *tag*, co umożliwi odróżnienie przycisków i wykorzystanie instrukcji *switch*. Pozwoli to także na nadanie odpowiedniego koloru ustawionego wartościami *RGB* dla dostępnych pięciu przycisków na głównym widoku interfejsu aplikacji. Na przykładzie 12 przedstawiono implementację przycisków z kolorami.

Przykład 12. Implentacja kolorowych przycisków

```
- (IBAction)pencilPressed:(id)sender {
   UIButton * PressedButton = (UIButton*)sender;
   switch(PressedButton.tag)
   {
      case 0:
       red = 255.0/255.0;  green = 255.0/255.0; blue =
       0.0/255.0;
         break;
      case 1:
         red = 0.0/255.0;   green = 0.0/255.0;   blue =
        255.0/255.0;
         break;
   }
```

Zresetowanie wszystkiego, co zostało narysowane, zrealizowane zostanie poprzez przypisanie wartości *nil* dla obiektu *mainImage*. Metoda obsługująca przycisk *reset* przedstawiona została na przykładzie 13.

Przykład 13. Implementacja przycisku reset

```
- (IBAction)reset:(id)sender {
       self.mainImage.image = nil;
}
```

Wciśnięcie przycisku *save*, zapisze we wbudowanej aplikacji *zdjęcia* na urządzeniu to, co zostało aktualnie narysowane na obiekcie *UIImageView*. Jego implementację przedstawiono na listingu 14.

Przykład 14. Implementacja przycisku save

```
- (IBAction)save:(id)sender {
  UIGraphicsBeginImageContextWithOptions(self.mainImage.
  bounds.size, NO, 0.0);
  [self.mainImage.image drawInRect:CGRectMake(0, 0,
  self.mainImage.frame.size.width,
  self.mainImage.frame.size.height)];
  UIImage *eSaveImage=
  UIGraphicsGetImageFromCurrentImageContext();
  UIGraphicsEndImageContext();
  UIImageWriteToSavedPhotosAlbum(eSaveImage,
  self,nil, nil);
 }
```

4. Podobieństwa i różnice języków w aplikacji Magic Drawing Board

Obie aplikacje wykorzystują specjalny silnik *Quartz 2D* [6], służący do tworzenia dwuwymiarowej grafiki. *Quartz* bazuje na języku C, większość wywoływanych funkcji jest napisana w tym języku, lecz dla języka Swift wszystko jest napisane już w Swift. Rysowanie zawsze odbywa się w klasie *UIView* lub w klasach dziedziczących po *UIView*. Rysowanie bazuje na tworzeniu ścieżek, na których zostają namalowane linie oraz definiowane są kształty, które zostaną odmalowane. Funkcje operują na pojęciu punktu. Punkt to współrzędne X oraz Y ekranu, gdzie punkt 0,0 znajduje się w lewym górnym rogu ekranu. Wartości X i Y są przechowane w zmiennej typu *CGFloat*. Struktura *CGPoint* użyta w obu języka przechowuje te wartości dla punktów ekranu.

Wywołanie większości funkcji w *Quartz 2D* wymaga przekazania parametru tzw. *context* (kontekstu). Każdy widok posiada swój własny kontekst, który jest odpowiedzialny za przeprowadzenie operacji rysowania. Kontekst jest przechowywany w zmiennej o typie *CGContext* [7].

Można zauważyć, że w obu językach są zawarte takie same funkcje, gdzie ich nazwy zaczynają się od słowa *CGContext*, np.:

- *CGContextSetLineWidth;*
- *CGContextSetStrokeColorWithColor*;
- *CGContextMoveToPoint*.

Język Swift i Objective-C posiadają metody o takich samych nazwach: metody *touchesBegan* i *touchesMoved*. Są one metodami domyślnymi z automatycznie generowanymi parametrami. Warto zaznaczyć, że operowanie na obiekcie *CGContext* w języku Swift odbywa się w bardziej uproszczony sposób. Dla Objective-C wywołanie tych samych metod odbywa się bardziej precyzyjnie i szczegółowo. W języku Swift wystarczy tylko utworzyć zmienną, która wywoła funkcje *UIGraphics-GetCurrentContext*, następnie należy ustawić kropce wartości takich zmiennych, jak: *setStrokeColor*, *setLineWidth*, *move, addLine* itd. (listing 1). Zatem należy stwierdzić, że główną różnicą w obu przypadkach jest składnia, która godzi się lepiej na korzyść języka Swift .

Implementacja odpowiednich kolorów dla każdego z przycisków w każdym języku wygląda inaczej. W aplikacji napisanej w Objective-C został wykorzystany jeden z modeli przestrzeni barw *RGB*. Pozwoliło to na przypisanie odpowiednich wartości liczbowych dla każdej z trzech zmiennych: *red*, *green*, *blue*. W aplikacji dla języka Swift została wykorzystana specjalna do tego klasa *UIColor,* która zawiera w sobie piętnaście podstawowych kolorów. Oczywiście takie same rozwiązania można było by zastosować dla jednego i drugiego języka, ponieważ język Objective-C również zawiera taką klasę.

5. Podsumowanie

W artykule zaprezentowano metodyki tworzenia aplikacji mobilnych na platformę iOS w językach Swift i Objective-C. Na przykładzie stworzonej aplikacji wskazano podobieństwa i różnice podczas ich projektowania. Opisane zostały różnice między językami w implementacji różnych widoków od początku tworzenia projektu do jego końca.

W wyniku przeprowadzonych analiz napisanego kodu aplikacji można zauważyć, jak bardzo różnią się od siebie jezyki Swift i Objective-C. Zdecydowanie sprawniej można wykorzystać język Swift do implementacji różnych widoków. Tworzenie kodu w tym języku programowania jest bardziej zrozumiałe i przyjemne. W większości przypadków jest go mniej, a to jest ważną rzeczą dla programisty, ponieważ zachodzi mniejsze prawdopodobieństwo wyskoczenia błędu podczas kompilacji projektu. Różnicę tą można zauważyć na dwóch metodach implementujących dotyk i ruch rysowanej linii przez użytkownika. Już samo to, że klasa nie zawiera dwóch plików: interfejsu i implementacji, jak w przypadku aplikacji napisanej w języku Objective-C, stawia Swift na lepszej pozycji. Jedyną wadę, jaką można wskazć, to fakt,iż jest on w ciągłym rozwoju i składnia języka może ulec większej lub mniejszej zmianie. Jednak zwykle tego typu zmiany wychodzą na lepsze.

Język Objective-C nadal otrzymuje wsparcie od firmy Apple, lecz z biegiem czasu popularność języka Swift zdobywa coraz większe uznania wśród programistów tworzących oprogramowanie na platformę OS X i iOS. Czynnik ten może zdecydować, że w przyszłości przestanie on być używany.

Literatura

[1] code.tutsplus https://code.tutsplus.com/pl/tutorials/an-introduction -to-swift-part-1--cms-21389 [Dostęp: 14.12.2016].

[2] developer.apple https://developer.apple.com/xcode/ [Dostęp: 14.12.2016]

[3] Ng S.: Beginning iOS 9, Programming with Swift. AppCoda Limited. 2015.

[4] Ng S.: Intermediate iOS Programming with Swift. AppCoda Limited. 2015.

[5] Kochan S.G.: Objective-C, Vademecum profesjonalisty. Helion, Gliwice. 2012.

[6] developer.apple https://developer.apple.com/library/content/documenttationn/GraphicsImaging/Conceptual/drawingwithquartz2d/Introduction/Introduction.html [Dostęp 14.12.2016]

[7] code.tutsplus https://code.tutsplus.com/tutorials/an-introduction to-quartz-2d--cms-24267 [Dostęp 14.12.2016]